Performance Evaluation of Brute Force Techniques for Routing and Spectrum Assignment in Elastic Optical Network using MPI and CUDA

Mahmoud Fayez
Computer Systems Department
Ain Shams University
Cairo, Egypt
mahmoud.fayez@cis.asu.edu.eg

Iyad Katib
Computer Science Department
King Abdulaziz Univesity
Jeddah, KSA
iakatib@kau.edu.sa

George N. Rouskas
Computer Science Department
North Carolina University
NC, USA
rouskas@ncsu.edu

Tarek F. Gharib
Information Systems Department
Ain Shams Univesity
Cairo, Egypt
tfgharib@cis.asu.edu.eg

H. Khaled
Computer Systems Department
Ain Shams University
Cairo, Egypt
heba,khaled@cis.asu.edu.eg

H. M. Faheem
Computer Systems Department
Ain Shams University
Cairo, Egypt
hossam.faheem@cis.asu.edu.eg

***Abstract*— In Elastic Optical Networks, Routing and Spectrum Assignment algorithms find a routing table and assign spectrum slots for each traffic demand such that the consumed spectrum on each optical link is minimized. This work deploys brute force algorithm to investigate all possible routing tables and select best candidates. Those candidates are then evaluated using the Spectrum Allocation algorithm to select the optimum spectrum utilization on the links. Routing table selection process depends on a scoring function that can be computed in linear time. Starting with an initial routing table (depending on shortest path algorithm) as an initial seed, the Selection criteria of the routing table decide whether to keep the initial seed or replace it with the one that can lead to a much more better solution. This in turn will eliminate the need to solve the spectrum assignment problem for each routing table. Consequently, the time needed to solve the problem will be dramatically reduced. Parallel computing paradigms are used to implement the brute force algorithm on parallel architectures to speed up the processing time. Performance comparison between MPI implementation and CUDA implementation is presented which shows that single GPU can perform better than high-end set of servers.**

Keywords—EON, RSA, CUDA, MPI, Brute Force

I. INTRODUCTION

Elastic Optical Network (EON) is getting more attention as it has a huge potential to support the increasing demand for data communication and telecommunication. Elastic optical spectrum can optimally support 400 GB/s and 1Tb/s. It requires a lot of changes to the optical network components [1] such as Reconfigurable Optical Add-Drop Multiplexer (ROADM) and Bandwidth Variable Transceivers (BV-T) that are be able to generate Elastic Optical Paths (EOPs). The continuous improvement in such network components is opening up exciting and promising new fields in optical network research.

The need to enhance the spectrum utilization of the C-Band (wavelength 1530 to 1565 nm) is pushing the optical networks towards elastic network[1]. EON is facing a lot of challenges; one of them is the Routing and Spectrum Assignment (RSA) problem. One of the common RSA problems is the selection of EOP which depends mainly on both the traffic demand's bit-rate and the distance between the source and the destination. Another challenge is assigning a common spectrum for a message travelling across multiple links.

This article presents a brute force (BF) algorithm for finding the optimum routing table of a given network that has actual/estimated traffic demand matrix. The BF solves the RSA problem in offline/static mode. The goal is to find the optimum routing table (EONs) such that the required spectrum of each optical link is minimum. BF evaluates all routing tables against a scoring function and picks the one with the highest score. The scoring functions used to evaluate those Routing Tables is the Spectrum Assignment Algorithm based on Longest First Fit (LFF) technique. BF also keep track of top performing Routing Tables based on the scoring function; so it accommodates link failure by providing alternative routing tables.

The rest of the paper is organized as follows, Section 2 will demonstrate the RSA problem categories and the progress made by researchers to address the static RSA problem. Section 3 will present the problem formulation. Section 4, 5, and 6 will demonstrate the Brute Force algorithm, MPI Algorithm, and CUDA algorithm respectively. Section 7 will show the results and our concluding remarks..

II. RELATED WORK

The Routing and Spectrum Assignment (RSA) problem is categorized into 2 groups which are Dynamic RSA and Static RSA. Dynamic RSA deals with already allocated optical paths and check if a new traffic demand can be accepted or not. Static RSA has a complete traffic demand matrix and start finding the optimum route for each traffic demand in order to minimize the spectrum on each optical link. The Static RSA problem is NP-hard problem[2]; and the optimum solution is difficult to find for large networks. The optimum solution cannot be better than the lower-bound which will be discussed in the next section. The lower-bound for the RSA problem varies depending on the routing table, i.e. routing table that directs all traffic demands through a certain optical link will cause the lower-bound to be very large. On the other hand routing table that make sure the traffic demands are balanced across links will cause a lot of spectrum fragmentation over the links.

Wan X et al [3] developed a dynamic RSA algorithm that solves RSA problem while handling different parameters like signal format, bit-rate, and spectrum bandwidth. The goal of the algorithm was to minimize the path length, and keep the spectrum continuity condition valid across all links without

overlapping. Klinkowski et al. [4] has proposed a solution to the static RSA problem with estimated traffic as integer linear programming problem (ILP); which is using a heuristic approach. It is difficult to characterize the performance of such heuristic approach. It does not scale to other problem variants [2]. S. Talebi et al. [2] have proposed 4 different scheduling techniques for the path network by transforming the problem into multiprocessor task scheduling. They developed some progress in problem transformation and utilized multi-core task scheduling algorithms to solve the SA problem. The best scheduling techniques they proposed are Longest First Compact Algorithm (LFC) and Widest First Compact Algorithm (WFC). Roza et al. [5] has proposed a metaheuristics approach to solve the Routing, Modulation, Spectrum Allocation (RMSA) problem with four different objective functions. They showed a near optimal solution for small networks which shows a promising solution that can be enhanced using the metaheuristics solution based on Tabu Search to support larger networks.

The multiprocessor task scheduling techniques have been in study for a decade. The transformation of the Spectrum Assignment (SA) problem to multiprocessor task scheduling problem has proven that SA problem is NP-Hard [2]. The similar characteristics of SA problem and Task Scheduling problem would allow utilizing schedulability analysis techniques [6], [7] to find out whether the current SA problem has sufficient spectrum on all links or not.

III. RSA Problem

RSA problem in Elastic Optical Network (EON) is defined as in (1), (2) and (3).

$$G = (v, A) \quad (1)$$

$$T = [T_{sd}] \quad (2)$$

$$O = [o_{sd}] \quad (3)$$

Where:

- ***G*** is a undirected graph,
- ***V*** is the set of nodes in the graph,
- ***A*** is the set of undirected arcs (Optical Links) connecting the graph nodes,
- ***T*** is the traffic demand matrix.
- ***O*** is the routing table matrix where each item in the matrix is an optical path corresponding to the source and destinations donated by the row and column indices respectively.

Each traffic demand from a source node ***s*** to a destination node ***d*** is assigned an optical path and contiguous spectrum based on the RSA algorithm. The assigned path is selected to minimize the total required spectrum used on any link. Each Optical Path (OP) must satisfy the following three conditions:

- Spectrum Contiguity Constraint restricts each traffic demand to be assigned to a contiguous spectrum.
- Spectrum Continuity Constraint restricts each demand to be assigned to the same spectrum across all links of its path.
- Non-overlapping Constrains requires that traffic demands that share the same link to be assigned non-overlapping spectrums with guard band between them.

There are multiple possible paths from source node s to destination node ***d***. The number of possible paths $l_{s,d}$ varies depending on the topology and can be obtained using depth first search (DFS) or k-shortest paths algorithms. Choosing the shortest path for all traffic demands would result in converting the problem to a smaller special case problem called Spectrum Assignment (SA)[8], [9].

The problem can be divided into a set of sub problems and each one can have one or more solutions that can be used as input to the next sub problem. The RSA problem can be divided into the following sub problems:

- Find the routing-table based on the routing algorithm.
- Choose the modulation for each traffic demand based on the bit-rate and the distance.
- Define the actual spectrum slots and the optical path for each traffic demand that satisfy the 3 conditions mentioned before.

In this work we are focusing on solving sub problems 1 and 3 and the sub-problems are treated separately. We are comparing the lower-bound of the routing table against the best make-span to make sure the solution of the first sub-problem produce good candidates (inputs) for the next sub-problem.

A. *Solution Quality (Make-Span)*

Each optical link may be included in one or more optical paths. Those optical paths are used to fulfill certain traffic demands with certain bandwidth requirements. The required spectrum on that optical link will be the total sum of all traffic going through the optical link along with the guard bands. This is the lower-bound which is considered the optimum solution that may be achieved. The lower-bound not necessarily means it can be achieved. Make-Span on the other hand includes the fragmentations in the spectrum of the optical link due to the EON constraints that we discussed before.

Comparing two routing tables based on make-span will be wrong as this not necessarily the optimum solution due to the fact that the scoring function is using heuristics algorithm. Therefor comparing the best known make-span based on the best know algorithm against all other routing tables lower-bounds to find any routing table that may lead to better make-span is the right decision. The lower-bound is calculated for each routing table and is compared with the make-span of the best solution in hand, if the lower-bound indicates a possibility to have a better solution the sub-problem 3 is then solved for this routing table otherwise this routing table will not be further processed and it will be ignored.

IV. Brute Force Algorithm

The algorithm requires the input adjacency matrix ***A*** to be symmetric and the links to be designated by a sequence starting from one. i.e. a four node complete mesh is represented in (4) and (5) which has 6 links as shown in Figure 1. If there is no link the corresponding element of the matrix will have zero value. BF calculates all the possible acyclic paths from any source to any destination and store it into a matrix called *L*, i.e. a four nodes complete mesh paths matrix is shown in (6).

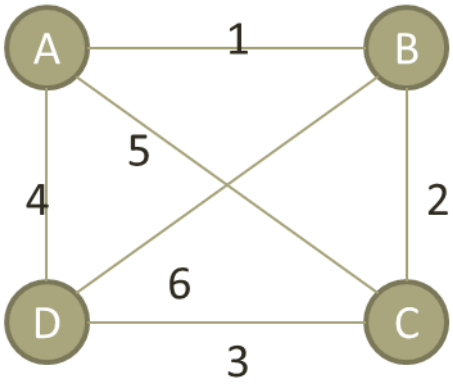

Fig. 1. Example of 4-node complete mesh with the links labeled starting from 1.

$$A = \begin{bmatrix} 0 & 1 & 5 & 4 \\ 1 & 0 & 2 & 6 \\ 5 & 2 & 0 & 3 \\ 4 & 6 & 3 & 0 \end{bmatrix} \quad (4)$$

$$v = \{A, B, C, D\} \quad (5)$$

$$L = \begin{bmatrix} l_{A,A} & l_{A,B} & l_{A,C} & l_{A,D} \\ l_{B,A} & l_{B,B} & l_{B,C} & l_{B,D} \\ l_{C,A} & l_{C,B} & l_{C,C} & l_{C,D} \\ l_{D,A} & l_{D,B} & l_{D,C} & l_{D,D} \end{bmatrix} \quad (6)$$

$$l_{A,B} = \{1, 18, 40, 52, 14\} \quad (7)$$

$l_{s,d}$ is a set of acyclic paths between Node *s* and Node *d*. Each acyclic path can be represented as a Boolean array where the i^{th} element of the array represent whether this link will be included in the optical path or not. This Boolean array would take too much memory in case of generating all possible paths so it has been presented as a set of M-bit integers where M is the total number of links in the graph. If the i^{th} bit is set this means the corresponding link will be used to transfer data between s and *d*.

The optimum representation of the path lead to optimum memory footprint of the algorithm such that we can generate all optical paths for a network with 100 links using less than 256 GB of memory. The numbers that are presented at (7) are calculated based on Table 1 which shows how the paths are converted to binary value of length M-bits.

TABLE I. Possible Paths from Node A to Node B

Path	Required Links	Binary Encoded						Decimal Value
		6	5	4	3	2	1	
A -> B	1	0	0	0	0	0	1	1
A -> C -> B	5,2	0	1	0	0	1	0	18
A -> D -> B	4,6	1	0	1	0	0	0	40
A -> C -> D -> B	5,3,6	1	1	0	1	0	0	52
A -> D -> C -> B	4,3,2	0	0	1	1	1	0	14

The L matrix is calculated using a Depth First Search (DFS) Algorithm; another algorithm like k-shortest paths can be used but it would be biased to the shortest path solution. It is assumed that the route between any two nodes will be the same for both traffic directions. i.e route from A to B and from B to A are the same. This assumption reduced the search space significantly. The DFS algorithm is used in two cycles the first cycles is used to count the number of the required paths and allocate enough memory for those paths. The second cycle of DFS builds the L matrix into the memory. Finally the BF algorithm iterates over all possible routing tables and find the lower-bound for each RT, RT whose lower-bound is less than the best make-span we have will be further processed and its make-span is calculated using LFF technique. If the new make-span is better, the list of the best RTs is updated. The pseudo code is shown in Figure 2.

```
1. RSA Program
2. Input: A the adjacency matrix of the network nodes.
3. Input: T the actual/expected traffic demand matrix in terms of how
many slots.
4. Output: O[] array of optimum routing tables sorted by the scoring
function.
5. Begin
6.    Call Provision All Paths(A) → L
7.    For i 1 → N
8.       For j 1 → N
9.          RT(i,j) ← L(i,j,0)          !select the first path from each set as
initial seed.
10.         RT_idx(i,j) = 0
11.      END FOR
12.   END FOR
13.   While (RT IS NOT NULL)
14.   BEGIN
15.      IF (LowerBound(RT) < BestScore) THEN
16.         Score = LFF(RT)
17.         IF (Score < BestScore) THEN
18.         BEGIN
19.            Call InsertRTInPlace(RT, Score) → BestScore;
20.         END
21.   ENDIF
22.   CALL IncrementRT(0, 1, L, RT, RT_idx, BestScore) → (RT,
RT_idx)
23.   End While
24.End
```

```
25.Subroutine IncrementRT
26.Input: i the source index
27.Input: j the destination index
28.Input: L all possible paths matrix
29.Input: RT the current routing table.
30.Output: RT next routing table
31.BEGIN
32.    IF (j > N) THEN
33.        RT ← NULL
34.    ELSE IF (i > j) THEN
35.        IncrementRT(0, j+1, L, RT)
36.    ELSE
37.        RT(i,j) ← ( RT(i,j) + 1) % length(L(i,j))
38.        IF (RT(i,j) == 0) THEN
39.            IncrementRT(i+1, j, L, RT)
40.        ENDIF
41.    ENDIF
42.END
```

Fig. 2. Pseudo code for BF

The score is calculated based on the Longest First scheduling technique that was presented in a previous paper by Fayez et. al [8]. Other scoring functions like Widest First Fit (WFF) can be used but as the results shown in the previous work Longest First was performing better than the Widest First technique. WFF may perform better under certain conditions. So better solution would be calculating the score of each RT (that has good lower bound) using all possible techniques but this would add more layer of complexity and would increase the time required to process all RTs in the search space.

V. MPI Algorithm

The brute force algorithm takes many hours as shown in the results section. So we had to parallelize it using MPI. The search space is divided among the MPI ranks evenly. Each MPI rank start provision all possible paths ***L***. Instead of waiting for master rank to calculate ***L*** and broadcast it to other ranks. This eliminates the need to broadcast ***L*** to all other ranks. Each MPI rank start with different initial seed as shown in **Error! Reference source not found.** (lines 13-15). There is not communication among the ranks as each rank produce different output file. Python script is used to merge the output of the MPI processes later as a post-processing step and the results section do not include the post-processing time as it is insignificant.

Each MPI rank will stop when it reaches the start offset of the sub-space of the next MPI rank. As it is shown in **Error! Reference source not found.** (line 35).

```
1. RSA Program_MPI
2. Input: A the adjacency matrix of the network nodes.
3. Input: T the actual/expected traffic demand matrix in terms of how many slots.
4. Output: O[] array of optimum routing tables sorted by the scoring function.
5. Begin
6.    Call Provision All Paths(A) → L
7.    For i 1 → N
8.       For j 1 → N
9.           RT(i,j) ← L(i,j,0)          !select the first path from each set as initial seed.
10.          RT_idx(i,j) = 0
11.      END FOR
12.   END FOR
13.   Offset ← L(N-1, N).Size() / MPI_Size * MPI_Rank
14.   RT(N-1,N) ← L(N-1,N,Offset)
15.   RT_idx(N-1,N) = Offset //Update the initial seed of last entry of RT to point to correct sub-space based on MPI Rank
16.   While (RT IS NOT NULL)
17.   BEGIN
18.      IF (LowerBound(RT) < BestScore) THEN
19.         Score = LFF(RT)
20.         IF (Score < BestScore) THEN
21.         BEGIN
22.            Call InsertRTInPlace(RT, Score) → BestScore;
23.         END
24.   ENDIF
25.   CALL IncrementRT(0, 1, L, RT, RT_idx, BestScore) → (RT, RT_idx)
26.   End While
27.End
28.Subroutine IncrementRT
29.Input: i the source index
30.Input: j the destination index
31.Input: L all possible paths matrix
32.Input: RT the current routing table.
33.Output: RT next routing table
34.BEGIN
35.    IF (j = N) AND (i = N-1) AND (RT(i,j) = N / MPI_Size * (MPI_Rank+1)) THEN
36.        RT ← NULL
37.    ELSE IF (i > j) THEN
38.        IncrementRT(0, j+1, L, RT)
39.    ELSE
40.        RT(i,j) ← ( RT(i,j) + 1) % length(L(i,j))
41.        IF (RT(i,j) == 0) THEN
42.            IncrementRT(i+1, j, L, RT)
43.        ENDIF
44.    ENDIF
45.END
```

Fig. 3. Pseudo code for BR with MPI

VI. CUDA Algorithm

GPU architecture is different from the regular processors. GPU consists of Stream Multiprocessors (SMs) which have many cores that executes the same instruction on different data. Branching is very slow on GPU as some cores will be idle in case their condition failed to enter the branch. So optimum algorithm for GPU should avoid loops and branches as much as possible. Another constraint is the physical limits of the GPU as maximum number of threads that can be deployed at a time is limited by hardware resources (registers and memory). So to deploy sufficient threads to calculate the lower-bound of all RTs we had to do it in batches. The host side pseudo code is shown in **Error! Reference source not found.** (lines 1-23). The device pseudo code is shown in **Error! Reference source not found.** (lines 24-37)

```
1. RSA Program_CUDA
2. Input: A the adjacency matrix of the network nodes.
3. Input: T the actual/expected traffic demand matrix in terms of how many slots.
4. Output: O[] array of optimum routing tables sorted by the scoring function.
5. Begin
6.    Call Provision All Paths(A) → L
7.    COPY TO GPU (L)
8.    BatchSize ← 10240 * 1024          !we will start 10240 block * 1024 thread per block
9.    B ← ∏_(i=1)^N▒(∏_(j=i+1)^N▒L(i,j) ) / BatchSize
10.   RT_Scores[i] ← ∞   where i ∈ [0,BatchSize]
11.   FOR i ← 1 to B
12.      CALL<<<10240,1024>>> BF_CUDA(i * BatchSize, L) → (RT_Scores)
13.      ! Each thread will process different RT in each batch and will overwrite the
14.      ! corresponding score in the RT_Score only if the new RT in the new batch has
15.      ! better solution.
16.   BEGIN
17.   END FOR
```

```
18.  QSORT(RT_Scores)
19.  SaveTop100(RT_Scores)
20.END
21.Subroutine BF_CUDA
22.Input: Offset the offset of this batch
23.Input: L all possible paths matrix
24.Output: RT_Scores next routing table
25.BEGIN
26.   RT ← Decode(block_idx, thread_idx, )
27.   LB ← LowerBound (RT)
28.   IF (LB < RT_Scores(thread_idx)) THEN
29.      Score ← LFF(RT)
30.      IF (Score < RT_Scores(thread_idx)) THEN
31.         RT_Scores(thread_idx) ← Score
32.      ENDIF
33.   ENDIF
34.END
```

Fig. 4. Pseudo code for BF with CUDA (Host and Device pseudo codes are presented)

VII. Results

The testing and benchmarking for the proposed implementations requires the set of the following inputs/resources:

- Traffic demand matrix, which was generated randomly with normal probability distribution for bit-rates [10, 40, 100, 1000] Gb\s. Any other probability distributions would not affect the proposed work as it is a brute force solution which solves all RTs score function if necessary.
- A network topology and we used complete mesh networks. Incomplete mesh networks have been benchmarked too to show how the problem size increases exponentially and how this affects the optimum make-span achieved.
- Set of compute nodes which has a common configurations for MPI testing. Each compute node consist of dual socket XEON processors each has 12 cores, memory of 96GB for each compute node. Each XEON core is running at 2.5 GHz.
- Single GPU node which has the same configurations in addition to NVidia GPU K20 which has 13 SM, each has 192 CUDA cores, total of 2496 CUDA cores running at 700MHz.

TABLE II. Results

Topology Size	Number of Links	Time (seconds) MPI 10 Compute Nodes	Time (seconds) CUDA 1 Compute Node
4-Nodes	6	1	2
5-Nodes	7	4	2
5-Nodes	8	5	6
5-Nodes	9	170	546
5-Nodes	10	7380	23280
6-Nodes	15	13 Years (estimate)	42 Years (estimate)

The time to find the optimum routing table for different topologies is presented in **Error! Reference source not found.**. The percentage of the routing tables that have been evaluated by LFF 0.000045%, which shows that most of the RTs will not lead to better solution using LFF. This shows how successful the skipping techniques is. The GPU algorithm shows that single compute node running with less than 750W can provide 33% of the performance of 10 compute nodes running at 3500W, the ratio of power between GPGPU consumption and MPI solution is 72% which means GPU is saving around 28% of the power. The power consumption is extracted from the manufacturer datasheets of the compute nodes.

Our future work will focus on skipping the need to calculate the complete Lower-Bound for each routing table and discard a group of routing tables based on their partial common lower-bound due to the fact that many RTs will have common routes.

References

[1] O. Gerstel, M. Jinno, A. Lord, and S. J. B. Yoo, "Elastic optical networking: a new dawn for the optical layer?," Communications Magazine, IEEE, vol. 50, no. 2. pp. s12–s20, 2012.

[2] S. Talebi, E. Bampis, G. Lucarelli, I. Katib, and G. N. Rouskas, "Spectrum assignment in optical networks: A multiprocessor scheduling perspective," Optical Communications and Networking, IEEE/OSA Journal of, vol. 6, no. 8. pp. 754–763, 2014.

[3] X. Wan, L. Wang, N. Hua, H. Zhang, and X. Zheng, "Dynamic Routing and Spectrum Assignment in Flexible Optical Path Networks," in Optical Fiber Communication Conference/National Fiber Optic Engineers Conference 2011, 2011, p. JWA055.

[4] M. Klinkowski and K. Walkowiak, "Routing and Spectrum Assignment in Spectrum Sliced Elastic Optical Path Network," Communications Letters, IEEE, vol. 15, no. 8. pp. 884–886, 2011.

[5] R. Goscien, K. Walkowiak, and M. Klinkowski, "Tabu search algorithm for routing, modulation and spectrum allocation in elastic optical network with anycast and unicast traffic," Comput. Networks, vol. 79, pp. 148–165, 2015.

[6] M. Marouf and Y. Sorel, "Schedulability conditions for non-preemptive hard real-time tasks with strict period," in 18th International Conference on Real-Time and Network Systems RTNS'10, 2010.

[7] O. Kermia and Y. Sorel, "Schedulability Analysis for Non-Preemptive Tasks under Strict Periodicity Constraints," Embedded and Real-Time Computing Systems and Applications, 2008. RTCSA '08. 14th IEEE International Conference on. pp. 25–32, 2008.

[8] M. Fayez, I. Katib, G. N. Rouskas, and H. M. Faheem, "Scheduling-Inspired Spectrum Assignment Algorithms for Mesh Elastic Optical Networks," Adv. Comput. Commun. Networks From Green, Mobile, Pervasive Netw. to Big Data Comput., p. 225, 2016.

[9] M. Fayez, I. Katib, G. N. Rouskas, and H. M. Faheem, "Spectrum Assignment in Mesh Elastic Optical Networks," in Proceedings of the 24th International Conference on Computer Communications and Networks (ICCCN), 2015.